A COMPUTER PROGRAMME WHICH LEARNS TO PLAY TICTACTOE

A Thesis

presented to the Department of

Electrical Engineering in partial

fulfilment of the requirements for

the Degree of Master of Technology.

By

Srinivasachar Murali

Master's Thesis EE-9-66.

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR.

APPROVED H. M. Nakabale .....(Thesis Supervisor).

ACKNOWLEDGEMENT

The author wishes to express his sense of deep gratitude to Dr. H.N.Mahabala for his constant guidance from the initiation to the completion of this thesis. Special thanks are also due to the staff of the Computer Centre for their invaluable help during programming; their forbearance and willingness to run the programme at all hours of the day and night is greatly appreciated.

CONTENTS

FIGURES

TABLES

FLOW CHARTS

CHAPTER ONE INTRODUCTION

1.1 The aim of artificial intelligence research is to produce human-like behaviour outside a living organism. It is motivated by a desire to understand the mechanism of thinking and learning in human beings. There exists no general statement of the problem of artificial intelligence. Research efforts in this field have been mainly directed towards solution of specific problems like game playing, theorem proving, pattern recognition and simulation of cognitive processes. A human being performing these tasks would be called 'intelligent'; any artifact which performed the same tasks, either in the same fashion as humans do or incorporating different information processing techniques would be called "artificially intelligent". These artifact may be mechanical or electromechanical systems (examples of which will be mentioned later on), hardware comprising of electronic components, or as is largely the case today, digital computers programmed to perform these tasks. Computer simulation of human behaviour is popular for the following reasons:

a) its high speed computational and symbol -manipulation abilities;

b) its vast 'memory' , where information can be stored and retrieved when needed, and,

c) its versatility-a general purpose computer can be made to perform a variety of specific tasks by simply providing it with a suitable programme. Also, a computer programme represents the researcher's

hypotheses about the processes underlying the behaviour. As such, it can be quickly tested by running the program on the computer and modified, if necessary. This process can be repeated till a satisfactory solution is obtained.

1.2 Persons trying to simulate various aspects of human behaviour on a computer using the technique mentioned above form one group of artificial intelligence researchers. There is a second group which wants to find new and better uses of computers by expanding the intellectual capacity of the machine, using whatever means are applicable. They are interested in creating computer programmes which manifest the kinds of behaviour we want, though it may not adopt human-like processes and techniques. There is a third group which believes that research in artificial intelligence can lead to better uses of machines by men than are being made today. Instead of the natural division of labor between a man and the machine - most routine work done by the machine and all generalisation and higher-level thinking by man - they believe in more cooperation and adaptive participation from machines in attacking complex and difficult problems. Their emphasis is on close interaction between man and machines.

1.3 An early attempt to model some aspect of human behaviour was made by Ross Ashby (Ashby, 1960). In his book 'Design for a Brain' he describes the design and construction of an electromechanical device called the homeostat. This device models the adaptive behaviour of the human nervous system.

Grey Walter's "M.Speculatrix" in a model of elementary reflex behaviour found in humans (Walter, 1961).

1.4 Probably the first serious attempt to simulate human theorem proving behaviour on a computer is a computer programme called the Logic Theorist developed in 1956 by Newell, Shaw and Simon on the RAND JOHNIAC computer. Although the programme could not model a brilliant logician, it made a significant landmark, in the sense that it helped psychologists and computer scientists to gain deeper insight into problems and techniques related to understanding human behaviour. It also started a whole new trend of symbol manipulation and so-called heuristic programming. Briefly, a heuristic programme is one which incorporates a rule of thumb, strategy, trick or any other kind of device which drastically limits search for solution to large complicated problems. Heuristics do not guarantee a solution to a problem; in this regard they are different from algorithms which always will lead to a solution. But where algorithms are impractical as in the case of complex problems, like chess-playing for example, heuristic methods offer more general applicability. The Logic Theorist was used to discover proofs to theorems in symbolic logic. It is interesting to note that the Logic Theorist gave rise to the first list-processing computer language, called the Information Processing Language or IPL.

The work of Gelernter (Gelernter, 1963) extends heuristic programming ideas to proofs of theorems in Euclidean geometry.

Feigenbaum's EPAM program, or the Elementary Perceiver and Memorizer, models verbal learning behaviour. It simulates the information-processing activity underlying human association learning.

Newell's Chess-machine and Samuel's Checker playing programme are examples of the fact that programmes can be written to exhibit 'learning' behaviour, though internally they contain no human-like problem-solving and learning processes. The programmes are able to improve their performance as they gain play experience . Samuel's Chekker Programme makes use of a linear polynomial expression with ingenious ways of selecting and changing its terms and of determining and modifying corresponding coefficients. The terms represent characteristics of game situation such as mobility, advancement and centre control.

Besides therem-proving and game playing, attempts at the solution of real problems have also being made. A good example is Tonge's Assembly-line balancing programme (Tonge, 1963). This is an example of application of heuristic programming to an important management science problem. Balancing and assembly-line involves finding and efficient arrangement of workers, tasks and work stations so as to maximise the rate of assembly or minimize the number of workers needed.

The above efforts represent attempts at solutions to specific problems only. These programmes cannot be used under varying circumstances- for example, the checker - playing programme cannot be made to play chess or even tictactoe.

aspects of intellectual behaviour is that investigation and evaluation are made easier. After enough information about different aspects of behaviour is accumulated ,some common features may be abstracted which could be used in more general problem-solving situations. Also, it is preferable to concentrate on particular problems if artificial intelligence research is to be of use in solving some immediate problems.

There have been attempts toward construction of more general intelligent systems. The General Problem Solver or GPS developed by Newell, Shaw and Simon is the first large-scale attempt of this kind. They used their earlier experience with the Logic Theorist in the design of GPS; some of the common features observed to be generally useful heuristics have been abstracted, and these form the 'general-part' of GPS, separate from the problem-specific part of the programme. GPS can handle a variety of problems like solving trigonometric identities, compile computer programmes and to prove theorems in the propositional calculus. The most extensively used technique in GPS is the 'means-ends analysis'. An initial problem state is transformed into the desired target state by selecting and applying operations which, step by step, reduce the difference between the two states.

Hormann's computer programme called 'Gaku' is another attempt toward a more general system. In Gaku, the learning mechanisms form the important part; Gaku can find solutions to progressively more difficult 'Tower-of-Hanoi' problems using its previous experience. Gaku and GPS will be treated in some detail in a subsequent chapter.

1.5 Some of the current difficulties faced in this field of research lie in trying to find out how computers can learn new heuristic metods and rules and how the induction capability found in humans can be mechanised. The most important problem is in overcoming the man-machine communication barrier. As Simon has indicated, the dilemma is that we can design more intelligent machines if we could communicate with them better but we could communicate with them better if they were more intelligent!

1.6 In this thesis we will be mainly concerned with the aspect of 'learning' in artificial systems. ~~The next chapter will be concerned with~~ Theories of Learning and some of the programmes for machine learning will be ~~discussed in chapter 3.~~ proposed. A Learning programme which 'learns' to play the game of tictactoe will be presented ~~in chapter 4.~~

CHAPTER TWO THEORIES OF LEARNING

2.1 In this chapter we shall briefly deal with the theory of learning and see how it is possible to exhibit intelligent behaviour in artificial systems, particularly in computers. Several definitions of the term "learning" have been proposed, each suitable for the particular physiological process that is being described or modeled. The learning process involves modification of present actions through past experience. But mere modification is not enough; for example, a seagull which follows ships in search of food is said to have "learned" how to find food. If, on seeing a ship, it always flew in the opposite direction, it would have had its behaviour modified, but would not be said to have "learned". (Humphrey, 1933). Therefore, behaviour modification that is useful to the organism can be called learning. Humphrey proposes a continuum of learning, starting from simple adaptation or accommodation at one end, and maze-learning, puzzle-box learning and ape-learning, in stages of increasing complexity, leading to human learning at the other end. Pavlov's conditioned response can also be placed in this continuum.

Conditioned response implies associative learning as displayed by Pavlov's dog. Salivation at the sight of food is the "unconditioned stimulus", and then, when the food is presented and a bell rings at the same time, the bell may be called "conditioned stimulus". Conditioning occurs when the response of salivation is excited by the bell alone (now called the conditioned response) without the presence of food.

Most of the modern difinitions of learning have been in terms of conditioning and the problem of learning is reduced to the problem of "reinforcement". Reinforcement has been defined thus :

> "Any stimulus which can increase the strength of a response when presented in close temporal conjunction with the occurrence of the response" (Deese, 1952).

Three definitions of learning will now be quoted. Guthrie's definition (Guthrie, 1935) reads:

> " Learning is the ability to respond differently to a situation because of past responses to the situation".

Hilgard's definition (Hilgard, 1948) reads:

> " Learning is the process by which an activity originates or is changed through reinforcement procedures as distinguished from changes by factors not attributable to reinforcement".

Thorpe's definition (Thorpe, 1950) reads:

> " The process which produces adaptive change in individual behaviour as the result of experience."

Cannon (Cannon, 1929) takes the view that learning is inevitable for survival and calls it the process by which the behaviour is, in some way, modified by external or internal changes towards the survival of the organism. This is, of course, only slightly different from Humphrey's viewpoint.

Separate definitions have been proposed for several types of learning like, "trial-and-error learning", "latent learning", and "insight learning". Learning may occur when

trial-and-error learning. Some organisms can learn without immediately manifesting their learning in a changed performance. This is latent - learning.

2.2 The effect of learning in a human being is an increased ability to adapt himself to new situations. The process of adaptation may be looked upon as a specific problem-solving process or directing to seek a particular goal. Given a particular problem-solving situation, a person utilizes his earlier experience in solving similar or related problems and gradually arrives at the correct solution. He may display occasional flashes of insight, thus speeding up the problem-solving process. How can this process be mechanised? Or more particularly, how can a computer be programmed to simulate this problem-solving behaviour?

2.3 A computer possesses some of the requirements that are necessary to achieve the above end. It offers a large memory- space where information can be stored, input - output facilities for communication with the programmer and hardware for actual information processing - symbol manipulation and computation. It only remains to tell the computer how to attempt trial solutions for a problem, store the results of these efforts and utilize them in solving new problems. This can be done by providing it with a programme incorporating these features. The programme may also tell the computer how to generalise problemsolving situations so that it is able to solve as large a class of problems as possible.

A computer capable of such generalisation can be said to possess the ability of inductive inference. This utilization of past experience is a form of feedback of information. Programmes utilizing this form of information feedback will be described in the next chapter.

CHAPTER THREE PROGRAMMING COMPUTERS TO LEARN

3.1 In this chapter, we shall briefly consider some of the computer programmes for learning that have been proposed. These programmes illustrate how computers can learn, utilising their previous experience in similar situations. We shall consider the following systems: Hormann's "Gakn" - the artificial student, Newell, Shaw and Simon's General Problem Solver, Oettinger's Learning Programme, Shannon's Chess Machine and Samuel's Checker Player.

3.2. Hormann's (1962, 1964) learning system incorporates some higher-order feedback features. Feedback allows a system to interact with the environment and correct its behaviour by coming nearer to its goal - as in a servomechanism. But there is no facility for recording and using its past experience; if the task were to be repeated, the same trial-and-error behaviour would be repeated before proper adjustment was made. Previous experience can be conveniently stored in the computer and utilized in order to avoid repeating the same mistakes.

Gakn is the Japanese word for learning. It consists of hierarchically interacting feedback-loop units, each of which is equipped with general rules for decision-making and information-handling within its prescribed domain of authority and activities.

It has four mechanisms, which are embedded in a master feedback unit called the mechanism coordinator. This mechanism coordinator, as well as each mechanism is designed

to apply a basic cycling process involving three phases (see figure 3.1). An analysis and test phase, a tentative selection or correction phase, and a consequence generation phase. A feedback loop is formed when the analysis and test phase receives the output of the consequence generation phase. Upon re-entering the analysis and test phase, reformulation of the given task is done by comparing the consequences with the description of the task. A fresh selection or a correction of the previous one is effected in the tentative selection or correction phase. The three phases are repeated until a success or failure is determined by the analysis and test phase.

This general structure is common to all four mechanisms and the mechanism coordinator. The four mechanisms have the different functions of planning, programming, problem-solving, and induction. The programming mechanism generates internal programmes from externally given description of the task. The planning mechanism divides the given task into sub-tasks which are easier to perform. These sub-tasks are solved by the problem-oriented mechanism. Efficient use of past experience is made using the induction mechanism which surveys the system's previous experience with similar tasks and applies it to new situations. Since humans are better than machines in planning and induction, in a partnership between man and this system, man can, through the mechanism coordinator give suggestion about new conjectures to be tried or subgoals to be set.

Gakn's performance has been tested using a sequence of problems from the "Tower-of-Hanoi" puzzle. Gakn was able

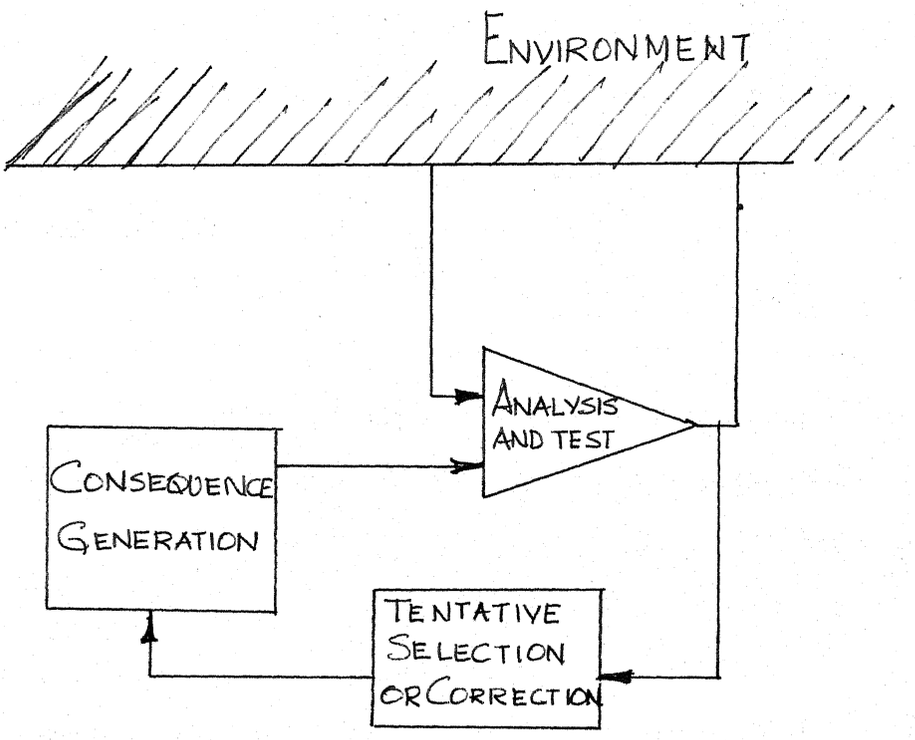
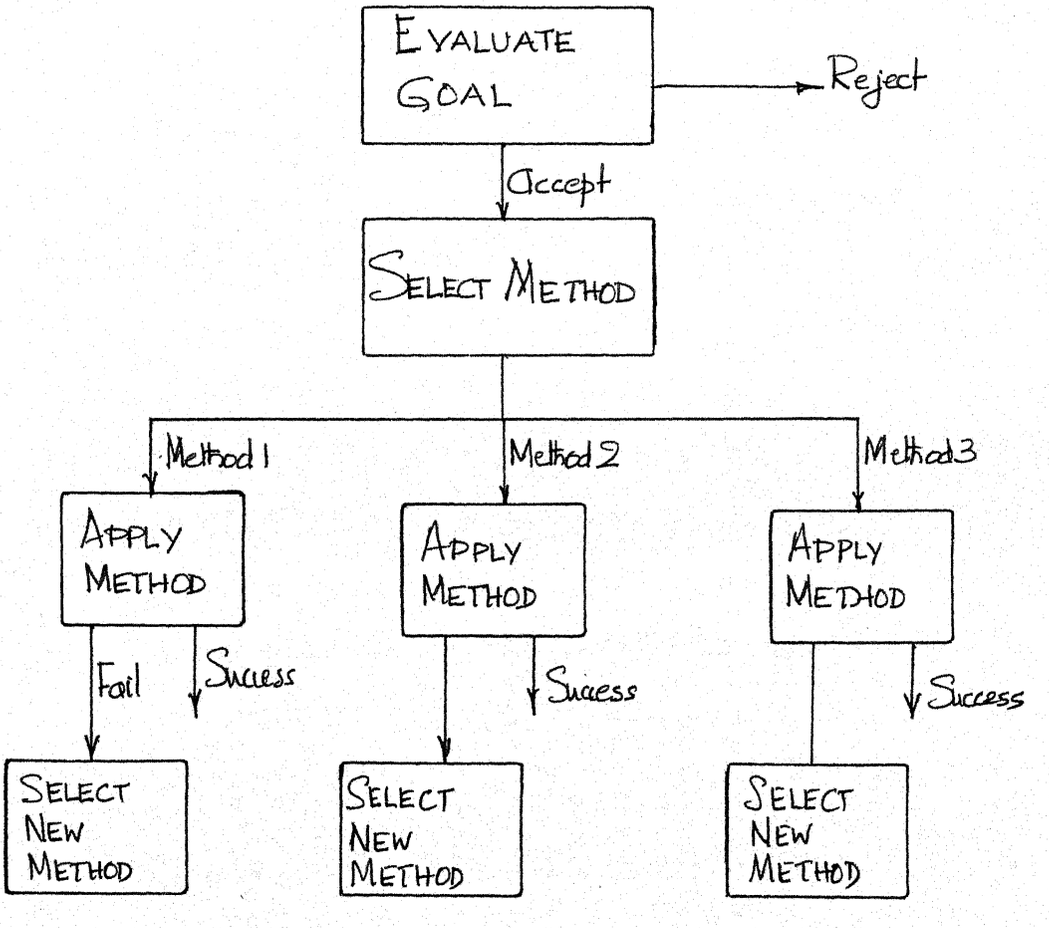


FIG 3.1: BASIC 3-PHASE CYCLE OF A GAKU MECHANISM

FIG. 3.2: BASIC ORGANISATION OF GPS.



to find solutions to progressively more difficult Tower-of-Hanoi problems, each time utilizing its past experience and finally finding a general solution ~~kxxkhm~~ pattern through its induction mechanism.

3.3 Learning possibilities in Newell, Shaw and Simon's (Newell, 1960) General Problem Solver or GPS have been analysed by them. As already mentioned, GPS is a programme that incorporates heuristic means for solving a substantial range of problems, for example, discovering proofs for theorems in logic, proving algebraic and trigonometric identities and performing formal integration and differentiation. A brief description of GPS follows:

GPS is a programme for working on tasks in an environment consisting of "objects" and "operators". Symbolic logic is one particular such task environment, in which GPS can operate. Here, the objects are logic expressions; the operators are the allowable rules of logic for transforming one expression to another. By successively operating on a given logical expression, it is simplified to the required form. But the object and operator are not confined to logic. GPS generally operates this way: it detects "differences" between objects and organises the information about the task environment to goals. There are three types of goals:

Transform object A to B,

Reduce difference D between object A and B,

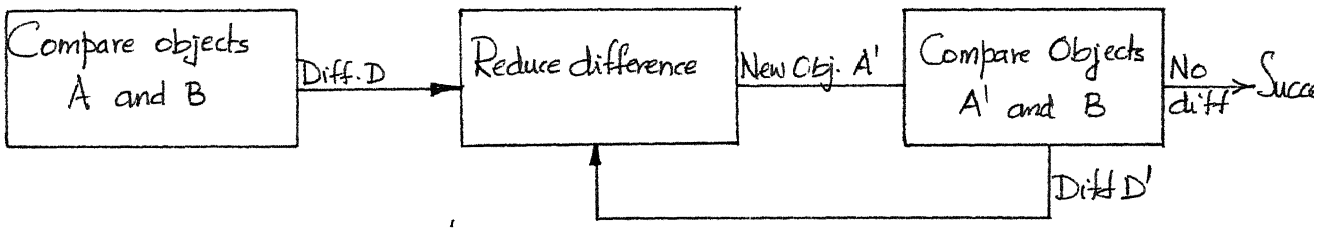
Apply operator R to object A.

The goals are achieved by setting up sub-goals, whose attainment

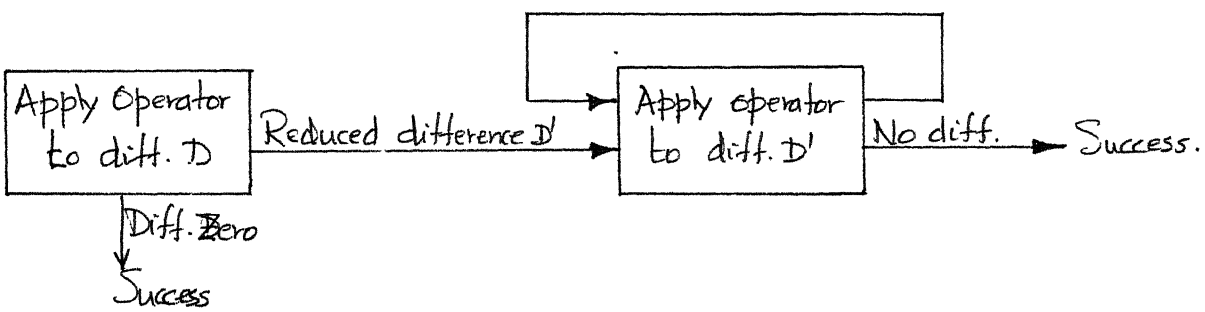
leads to the next subgoal to be achieved. The basic organisation of GPS for this purpose is shown in figure 3.2. It evaluates the goal to see whether it should be worked on; if it accepts the goal, it selects a method associated with the goal. If the method fails, it selects another method and applies it to achieve the goal. Figure 3.3 shows the methods employed for achieving the three goals. Thus, to transform and object A into an object B, the objects are first compared element by element. If a difference is revealed, then a subgoal is set up to reduce the difference. If this subgoal is attained , a new object will be produced which no longer has the same difference noted earlier. A new subgoal is now created which will further reduce the difference. This process is repeated until no difference is detected. The entire goal will then have been achieved.

Learning in GPS is achieved by having the learning situation shown in figure 3.4. A learning situation requires another programme called the Learning Programme. The performance programme at the bottom of the figure is GPS. The learning programme operates on the performance programme to produce a new performance programme better adapted to its task. GPS will thus become a better problem solver. The performance programme has two parts: (1) GPS proper, with its goal types and methods which are independent of the subject matter; and (2) the specification of the particular task environment: its objects, operators and its differences. A fuller description

Goal Type 1: TRANSFORM OBJECT A TO OBJECT B:



Goal Type 2: REDUCE DIFFERENCE BETWEEN A AND B:



Goal Type 3: APPLY OPERATOR R TO OBJECT A:

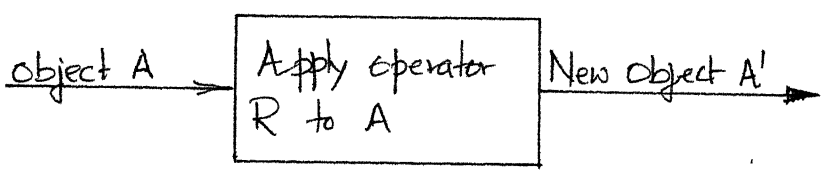


FIG 3.3: GOAL METHODS FOR GPS.

of the GPS and analysis of the learning situation can be found in the publication cited above.

3.4 Oettinger (1952) describes some techniques by which a computer can be made to display learning. He divided the computer into two parts, letting one part play the role of the learning machine and the other, the enviorment. He interpreted the learning activities as a series of shopping expeditions , where the shops are described by an mxn matrix the elements $a_{ij} = 1$ if shop i has article j, otherwise $a_{ij} = 0$. Thus, rows represent shop vectors and columns, article vectors. After an initial period of trial-and-error, the computer would know which shop contained a specific article and also what articles any given shop stocked. This displayed an elementary case of learning.

3.5 Most of the current day efforts however, have been in programming computers to learn to play games, particularly chess and checkers. The earliest scheme was proposed by Shannon (Shannon, 1950). He divided the problem into two parts: (1) Chosing a suitable code to represent the positions and pieces on a chess board as numbers; (2) chosing a strategy of play. The second part was the most difficult part of the problem, since the straightforward strategy of calculating all possible variations of all moves to the end of a game is beyond the capabilities of even present-day computers. In a typical chess position there will be about 32 possible moves with 32 possible replies-- already this creates 1024 possibilitie Most chess games last for about forty moves or more for each

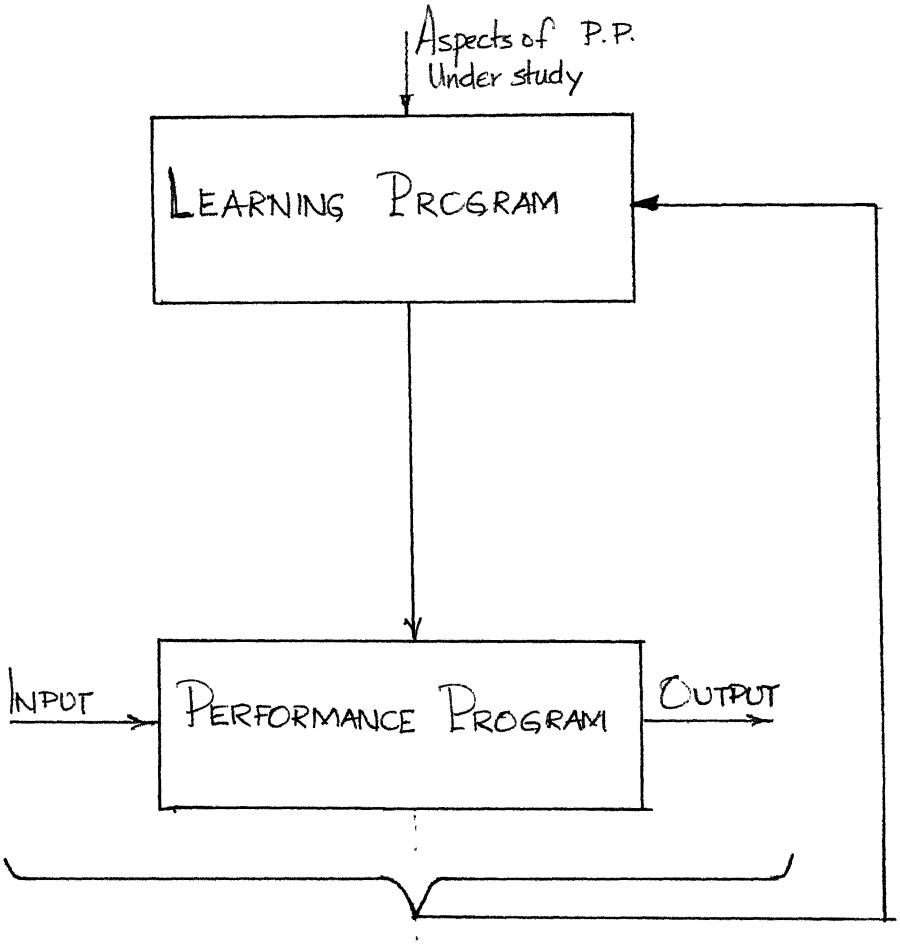


FIG 3.4: LEARNING SITUATION IN GPS.

side. The total number of possible variations in an average game is about 10^{120}. A machine calculating one move each millionth of a second would require many, many years to make its first move! So, Shannon used only few good strategies; his computer could play only a reasonably skillful game.

Shannon's programme did not possess any learning abilities. Newell (Newell, 1963b) has proposed a programme which incorporates learning. His machine works with a set of expressions or rules of thumb which provide it with the ability to play a reasonably good game. Following a sequence of games, it generates new expressions and modifies previous ones, thus evolving a set of expressions or rules of sufficient complexity to play good chess.

Many other chess programmes have been suggested and a specification and comparison of them could be found in Newell, Shaw and Simon (1963).

3.6 Samuel (1963) made extensive studies in machine learning using the game of checkers. He preferred checkers to chess because the simplicity of its rules permit greater emphasis to be placed on learning techniques. His computerm plays by looking ahead a few moves and by evaluating the resulting board positions, much like a human player. The board positions are "scored" in terms of its value to the machine with respect to advancement, mobility and centre control. Scoring is done in terms of a linear polynomial. Its terms and coefficients represent evaluation of the properties of various board positions mentioned above. Besides these, the parameters considered are relative piece advantage, number of

kings and their value in terms of pieces, and the inability for one side or the other to move which would decide the end of the game. Using these techniques, his machine learned to play a very good game of checkers.

3.7 In this chapter we have briefly considered several programmes for computers to learn. In the next chapter a specific learning scheme for a computer to learn to play the game of tictactoe will be presented.

CHAPTER FOUR A PROGRAMME WHICH LEARNS TO PLAY TICTACTOE

4.1 In this chapter, a computer programme which learns to play the game of tictactoe using a scheme of learning will be presented. The computer will gradually improve its ability to defend itself against the opponent. Learning will be said to be complete when computer no longer loses a game.

4.2 The Game:-

Tictactoe or, nought-and-crosses as it is also known, is a simple game played between two opponents. The tictactoe "board" or simply, "the board" as it will henceforth be referred to, consists of nine squares and the two players alternately mark a square with a nought (0) and a cross (X). The first player to have a row or a column or any of the two diagonals filled with his symbol (either a nought or a cross) is the winner. Game theory would describe this game as a two-person, zero-sum game. And, both players can adopt strategies which will always lead to a draw. Thus, our criterion of successful learning by the computer will be its ability to draw the game against any player.

4.3 Modus Operandi:-

(a) The first thing is to number the squares of the board, using the numbers one to nine inclusive. The choice of numbers is important because it directly affects the statement of a winning position. There are obviously many ways of carrying out this numbering, but this teems out to be a matter of no importance since the computer, provided it is told which game is won and which lost, will still be

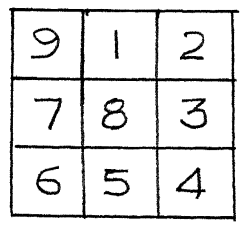
able to learn to play the game, regardless of the numbering system used. The numbering used in the present case is shown in figure 4.1. The reason for choosing this scheme was that we plan to make the first two moves of every game compulsory. This was necessary in order to decrease the demand on the computer memory space.

(b) Having decided upon the kind of numbering to be used, we must next provide the computer the ability to discover whether a game is won, lost or drawn. For this purpose, we have the following arrangement:

Each time the opponent selects a square, that square will be given a weight 1. The square selected by the learning programme will be given a weight 4. It is clearly seen that if the sum of any row or column or diagonal squares equals 3, the learning programme has lost; if any sum is 12, the learning programme has won; if neither, and all squares are full, the game is a draw.

(c) Assuming that the computer is given instructions as to how to select the squares, it must now learn that certain ways of carrying this process of selection lead to a win, others to a loss. In order that it will be possible to decide whether a move is good or bad, a complete record of the set of moves making up a game should be kept. There are nine or less moves in every complete game; the computer should be able to store all sequences (of different "lengths") arising from all possible combination of the numbers 1 through 9. There are more than 100,000 such sequences; the computer does not have such a large number of storage locations. Therefore, to reduce the number

of storage locations required the first two moves of every game are fixed to start with. Thus, the centre square and the top left hand square (Squares numbered 8 and 9 in figure 4.1) will always be the first two moves of every game.



TICTACTOE BOARD

Figure 4.1

The rest of the squares are numbered 1 through 7. There are 13,700 sequences possible. All sequences are initially given a "tactical value" which is positive. If a particular sequence is discovered to have led to a loss, its tactical value will be immediately made negative; the computer will not make choices which will give rise to such sequences. Before making a choice, the computer will select a square and check the tactical value of the sequence formed by the numbers selected so far. If the tactical value is positive, it will "announce" that selection as its choice; if it is negative, it will make a fresh selection and check the tactical value of the new sequence. If the tactical value is again negative it will make yet another selection. Thus the computer will finally select a number which gives rise to a sequence with a positive tactical value.

a set of sequences which lead to a loss and these will have a negative tactical value associated with each. All the other sequences will have positve tactical value associated with them and the combination of numbers* giving rise to these sequences will be "safe" choices for the computer against that player. Assuming a regular pattern of play by the opponent, we have the following illustrations of games (the first move is always by the opponent), L standing for lost and D for drawn:

8-9-1-2-5-L

8-9-1-3-5-L

8-9-1-4-5-L

8-9-1-5-2-3-6-L

8-9-1-5-2-4-6-L

8-9-1-5-2-6-3-4-L

8-9-1-5-2-6-3-7-4-D

After this point the game remains constant if the opponent plays consistently. We may also note that the first six sequences now have a negative tactical value associated with each of them.

Exactly the same process occurs with different opponents, although it takes much longer to reach a stable state. The computer can easily perform the operations described, and it very quickly learns the important tactical steps, after which it never loses. Another important feature is that the computer does not have the same statement of tactics

---

* It should be noted that the names "digit", "number", "square", "selection","choice" are all used interchangeably; they all

as the opponent. The opponent plays deductively whereas the computer carries through the steps of checking the tactical values from the store and finding out whether earlier experience in a similar situation has been against it.

(d) The method of storage is considered next. Each of the 13,700 possible sequences will have its associated tactical value stored at a particular location. To find the address where the tactical value of a particular sequence which has appeared in a game is available is a problem in itself. For this purpose, a formula is developed which will calculate the correct address using a number making up the sequence and the order in which they appeared. We first have the following arrangement:

1. All sequences with only one digit have their tactical values stored in locations 1 through 7. (Since the first two moves are compulsory, our sequences will make up of numbers 1 through 7 only).

2. There are 41 sequences having only two digits. These have their tactical values stored in locations 8 through 49. And so on. Table I gives the corresponding areas in the storage where the tactical values for sequences of all lengths are found. See Table II for formulae to calculate the address.

Since we wish the computer to learn to play the game, its speed will be greatly affected by the slowness of its in-put-output as compared with its computation speed. The delay created by a human opponent's thought is relatively enormous. It was therefore found convenient to have the learning programme play against an "Intelligent Programme",

TABLE I

No. of digits in a sequence	Area of storage where the tactical values are found
1	1 to 7
2	8 to 49
3	50 to 259
4	260 to 1099
5	1100 to 3619
6	3620 to 8659
7	8660 to 13700

The sequences are stored in the following order, starting location 1, ending location 13700:

1	35	71	•	•
2	36	•	•	•
3	37	•	7654	•
•	41	76	12345	•
•	42	123	•	1234567
7	43	•	•	•
12	45	•	•	•
13	46	127	•	•
14	47	132	•	•
15	51	•	•	•
16	52	•	•	•
17	53	•	•	•
21	54	213	•	•
23	56	•	76543	•
24	57	•	•	•
25	61	•	•	•
26	62	•	•	•
27	63	765	•	•
31	64	1234	•	•
32	65	•	•	•
34	67	•	•	7654321

TABLE II

IIa. Formulae to calculate the addresses where the tactical values various sequences are stored (the addresses are identified by the symbol IR):

1. For sequences with one digit: IR=ID1 (see IIb).

2. For sequences with two digits:

IR = 7 + IP2, where

IP2 = 6(ID1-1) + ID 2.

3. For sequences with three digits:

IR=7+7.6+IP3,

IP3=5(IP2-1)+ID3 .

4. For sequences with four digits:

IR =7 +7.6+7.6.5+IP4 ,

IP4=4(IP3-1)+ID4 .

5. For sequences with five digits:

IR =7+7.6+7.6.5+7.6.5.4+IP5 ,

IP5=3(IP4-1)+ID5 .

6. For sequences with six digits:

IR+7+7.6+7.6.5+7.6.5.4+7.6.5.4.3+IP6,

IP6=2(IP5-1)+ ID6 .

7. For sequences with seven digits:

IR=7+7.6+7.6.5+7.6.5.4+7.6.5.4.3+7.6.5.4.3.2+IP

IP7=(IP6-1)+ID7 .

IIb. ID1, ID2 etc. are found thus: The seven digits are stored in a register as shown below:

J	1	2	3	4	5	6	7
ITAB(J)	1	2	3	4	5	6	7

TABLE II (CONTD.)

This is referred to as the TAB register in the actual programme. J is an index and this assumes the same value as the actual number chosen by either programme. Once a digit is selected as a move, it is 'removed' from the register and all the digits to the right of this digit are reduced by 1. For example, if 3 is the number selected first, ID1=3. The altered register would now look like this:

J	1	2	3	4	5	6	7
ITAB(J)	1	2	3	3	4	5	6

Instead of actually removing 3 by making it zero or by other means we have left it as it was in the register earlier. Since no digit is selected twice in the same game, this does not matter. If now, 5 in the next choice, then, ID2=ITAB(5)=4. The register will now appear thus:

J	1	2	3	4	5	6	7
ITAB(J)	1	2	3	3	3	4	5

This is repeated everytime a digit is chosen.

with the earlier moves of the game has a positive or negative tactical value associated with it. If it is positive, that choice is accepted. If negative, it will choose the next square and repeat the process. It will not make moves which have led to a loss in earlier games, after it has played a number of games. Its moves will be so as to lead the game to a draw. We then say that the Learning Programme has "learnt".

One other important aspect needs to be mentioned. Any game can be regarded as a tree, the nodes representing the "moves" and the branches, the "choice". For tictactoe, a tree would look like figure 4.2. $T_h$is does not represent any particular game of course, but only displays the several choices that become apparent at each move. After square 9 has been chosen, INTPRO has a choice of 7 square. For each square chosen by INTPRO, LEAPRO has six choices. And so on. Now, look at figure 4.3. $T_h$is is the complete tree for a game where INTPRO has chosen square 1 as its first choice. LEAPRO now has a choice of 6 squares and a choice of any of these squares except square 5 will lead to a loss for LEAPRO. According to its scheme, the LEAPRO will first play square 2, lose the game and make the tactical value for the sequence (1,2), negative. A similar fate will overtake sequences (1,3) and (1,4). Square 5 has now been discovered to be the safe choice. Suppose INTPRO now plays square 2. LEAPRO again has a choice of four squares and it will learn to play square 6 only after losing two more games by playing squares 3 and 4. As a result sequences (1,5,2,5) and (1,5,2,4) have negative

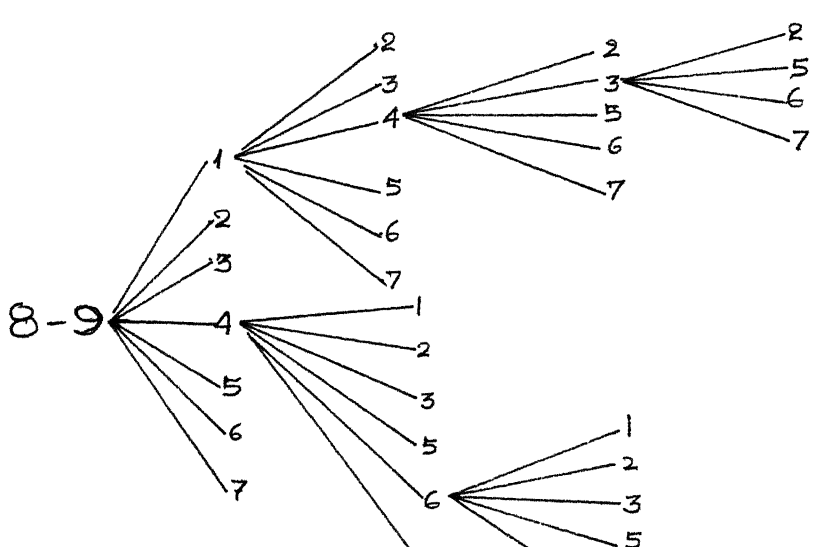


FIG 4.2 : TREE FOR TICTACTOE GAME.

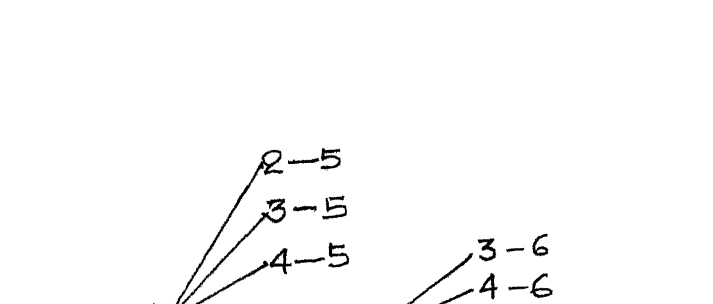


FIG 4.4 : TREE FOR GAME DESCRIBED ON PAGE 24.

tactical values. The INTPRO may next play square 3. If LEAPRO plays square 4, as it does the first time, it loses. Sequence (1,5,2,6,3,4) now has a negative tactical value. Square 7 is the safe choice and the game now ends in a draw. So far, altering of tactical values of "bad" sequences has not presented a problem. As soon as INTPRO wins a game, the Executor calls the LEAPRO and LEAPRO immediately changes the tactical value for the sequence ending in its move.

But supposing a situation as shown in figure 4.4a occurs. Such a position is called a "fork" and according to its scheme LEAPRO would play squares 1, 3, 5, 6, and 7 and lose all the time! Apparently for this game, any correction at this stage is of no avail, so one has to effect rectification at an earlier move. Apparently square 1 itself was a bad choice; so the sequence (4, 1) should have a negative tactical value. The LEAPRO should be able to detect such cases, go back and alter a previous choice. A tree for this game is shown in figure 4.4b. No satisfactory choice exists at node B. One has to go back to node A and make a different choice. For this reason, the LEAPRO is provided with two extra registers, where the addresses of previous sequences can be stored. These are identified as IR 1 and IR2, in the programme. Thus at any stage of the game, we have available to us the addresses of three different sequences, one corresponding to the latest move (in register IR), and two, corresponding to two previous moves (in registers IR1 and IR2). Once a fork has been detected, the LEAPRO will make the tactical value at the address stored in IR1 negative and instead of continuing the

4	4	1
	1	
		1

4.4(a): BOARD SHOWING 'FORK'

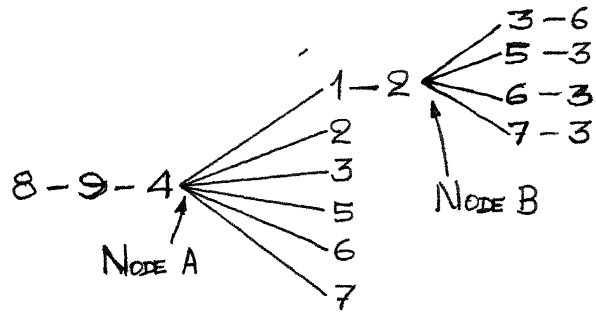


FIG 4.4(b): TREE FOR THIS GAME

4		4
1	1	4
1		

FIG. 4.6: 'FORK' IN FAVOR OF LEAPRO

2	1	9
3	8	7
4	5	6

(a)

6	5	4
7	8	3
9	1	2

(b)

4	5	6
3	8	7
2	1	9

(c)

9	1	2
7	8	3
6	5	4

(d)

FIG 4.7: ALTERNATE BOARDS.

game will concede it.

Flow Charts V, VI and VII show the subprogrammes ADRES, TAB and LEAST-which supplies LEAPRO with the least vacant square. These are sufficiently clear and need no explanation.

4.3 Programming details and results:-

(a) The programme was run on an IBM 7044 computer. The programming language used was FORTRAN IV, except for the RANDOM subprogramme which was coded in MAP, the internal language of the machine. A series of ten-thousand games were played between the INTPRO and the LEAPRO. Running time was about seven minutes including compilation time. The approximate size of the programme was:

1. Storage space for tactical values	13700 locatio
2. Storage space for the board, for storing the sums and for the TAB register	24 locatio
3. Total number of instructions	214 instruc-tions

(b) Firstly, the INTPRO was made to play against a random player, with no learning incorporated. Since the required random generator was already available, the Executor simply called the INTPRO and RANDOM subprogrammes alternately. It was discovered that the random player drew,on an average about five-per cent of the games it played. This formed a basis for comparison with the performance of the Learning Programme. The performance curve for the game between INTPRO and the random player is shown in figure 4.5a.

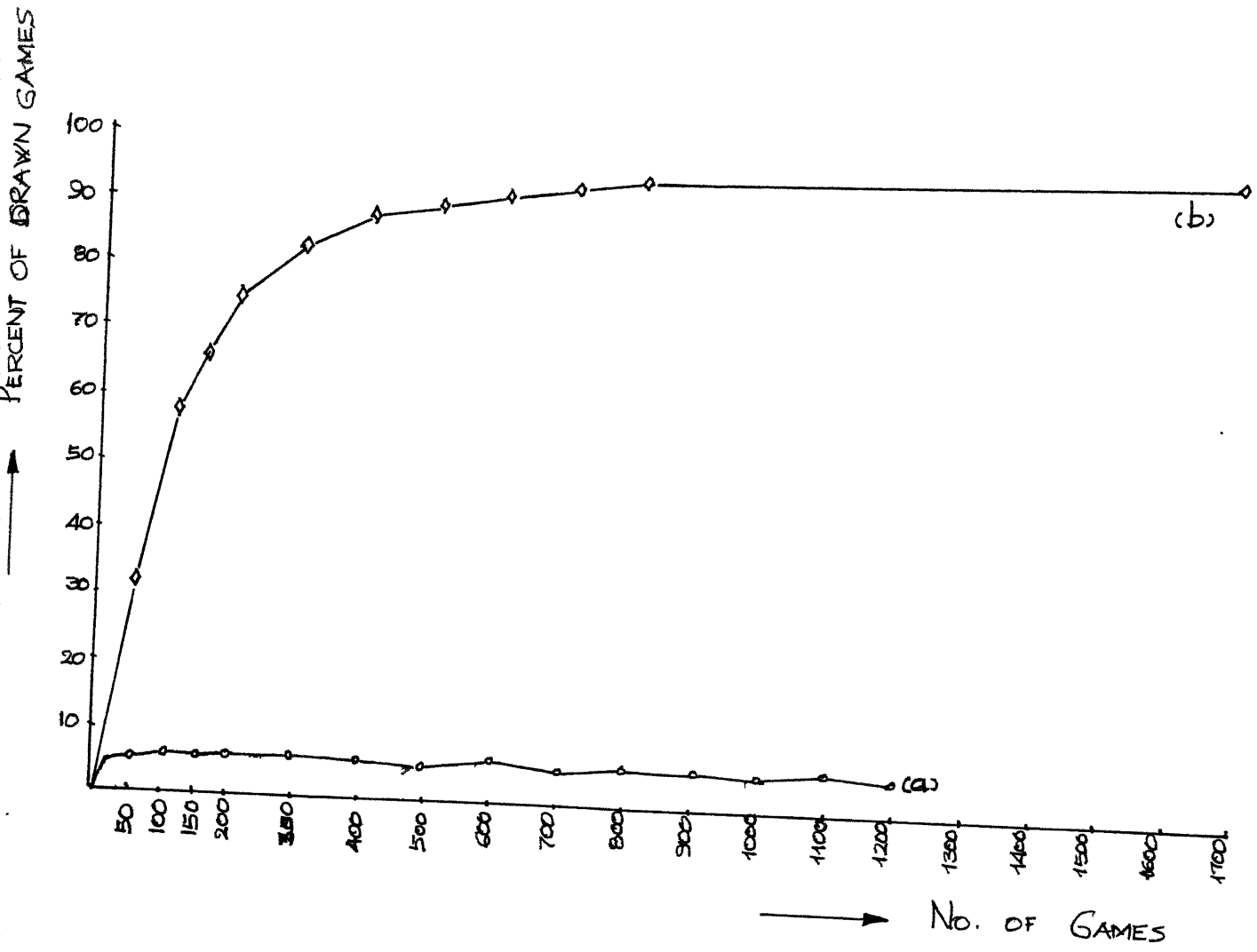


FIG 4.5 (a) AND (b): GRAPH SHOWING INTPRO vs. RANDOM AND INTPRO vs. LEAPRO.

(c) Next, the LEAPRO played a series of games with INTPRO. It drew its first game after losing the first twenty games. At the end of fifty games, it had drawn 16 and lost the rest. At the end of a hundred games, it had lost only 41 and drawn the rest - and encouraging improvement in its performance. After that, INTPRO could win only nine more games, thus making a total of fifty wins, after which it won no more games. Thus, steady state was reached after fifty games. Figure 4.5 shows the "learning curve" for this series of contests. A few words about learning curves are in order here. Since learning is an inference from performance, we must have some index for the measurement of performance. Learning curves are usually plotted with this index as one coordinate and time or number of attempts or any such similar variable as the other coordinate. In our case, the coordinates were, the games that were drawn, expressed as a percentage of the total number of games played, and the total number of games that were played. Learning curves are only data obtained in alleged learning situations arranged so as to quickly reveal a change in behaviour or performance. The manner in which the rise occurs is an important feature. The rise might occur with less and less improvement between trials or with more and more improvement. The sharply rising curves, according to psychologists, are supposed to reflect "insight" or some kind of sudden solution of a problem, the realisation of a significance, the perceptual

reorganisation of a field and so on (Guthrie, 1935). The curve shown in figure 4.5 shows a gradual rise which is typical of trial-and-error learning situations or learning from past experiences.

(d) An interesting situation which occurred in the course of these runs deserves mention. The following series of moves appeared: 7-3-6-2-1-4, where the first choice is by INTPRO. Clearly, this is a fork and in favour of LEAPRO! LEAPRO actually won this particular game and it was also discovered that this was the only game it ever won against INTPRO. Such situations are not many; this one was purely accidental. See figure 4.6 for the "board state" of this game.

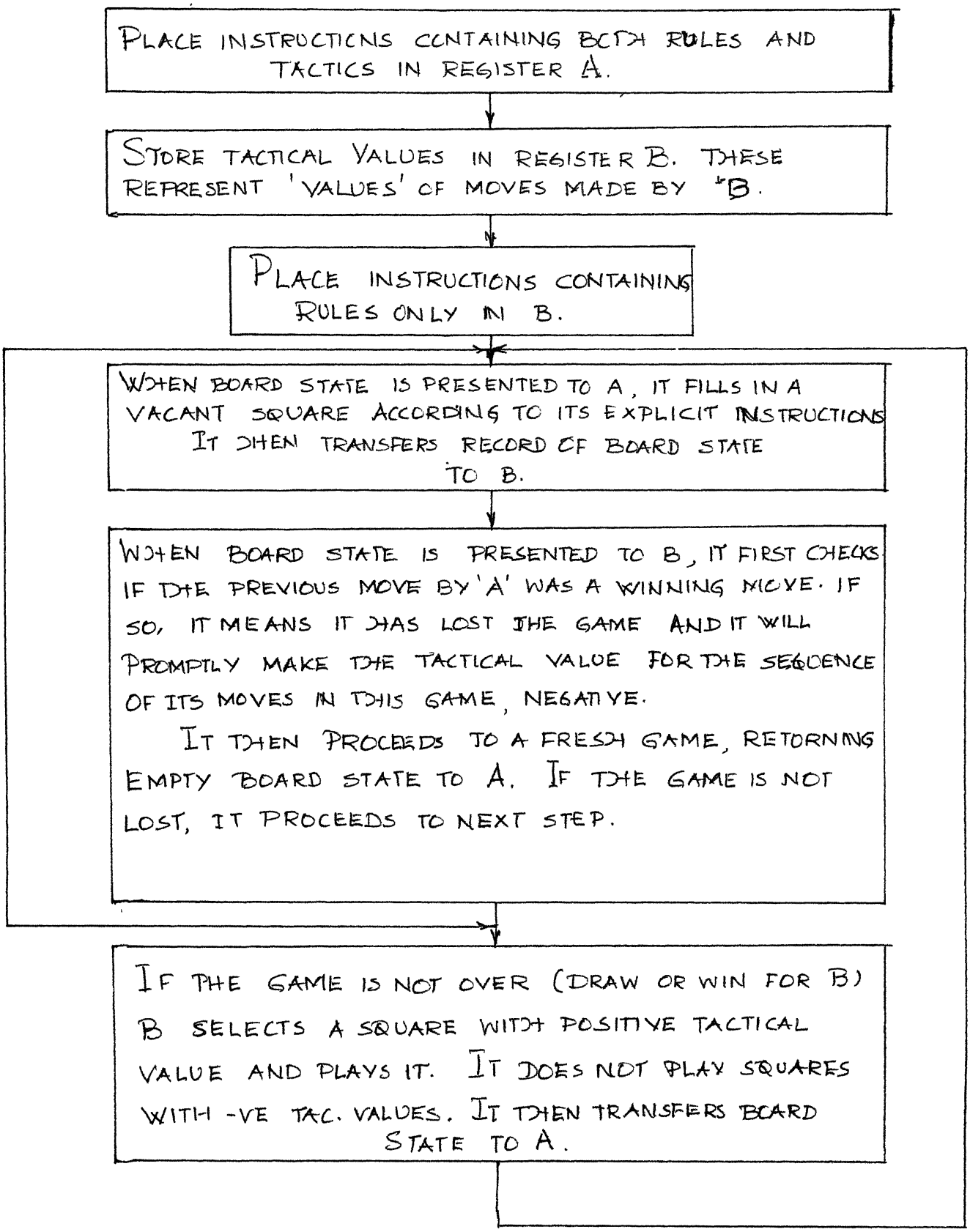
4.4 We have shown that the scheme of remembering sequences of moves and associating good (+ve) and bad (-ve) tactical values with them works for the game of tictactoe. Though only a limited number of games were played, it is clear that this scheme would work against any tactic by the opponent. The first two moves were restricted in order to reduce the demands on memory space required. A variation on this could be effected by changing the numbering of the board. For example, we could have the arrangement of figure 4.7a. In this case, we would only have to alter the definition of the row, column and diagonal sums. And, LEAPRO will continue to draw all the games. Or, we can have the alternates of figures 4.7b and 4.7c, and things would be the same. We can then say that

the Learning Programme has developed concepts of symmetry that are inherent in this game. Figure 4.7d shows the board used in the present case for immediate comparison with the other three figures.

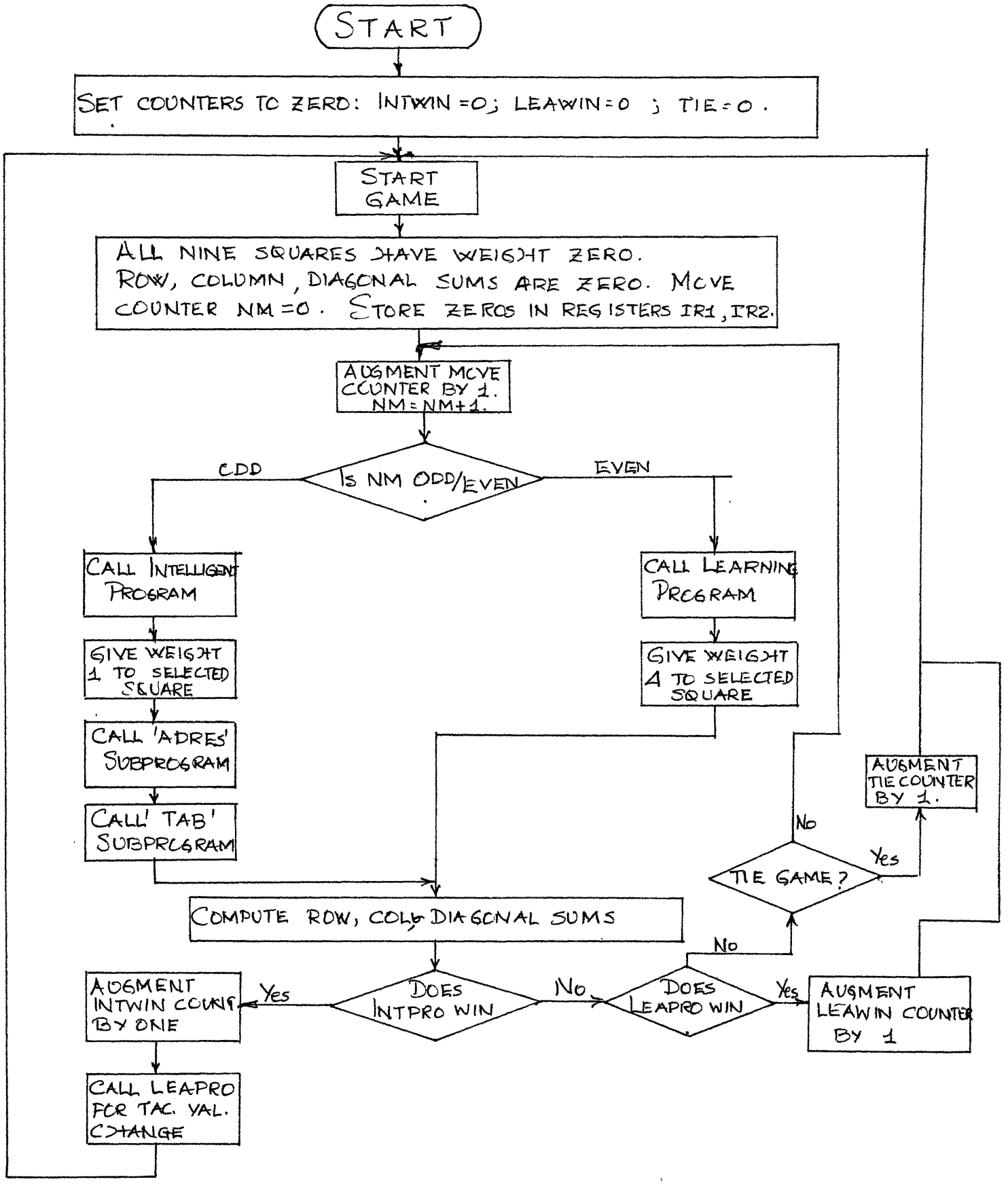
CONCLUSION

The fact that computers have been successfully programmed not only to play games like tictactoe, checkers and chess but also improve their performance with play-experience leads one to believe that "learning" or learning-like behaviour can be displayed by man-made systems. This is a significant step towards realisation of "artificial intelligence" - systems performing tasks, which if performed by man would be regarded as requring intelligence. There exists already, computer programmes which prove mathematical theorems, pattern recognition programmes, question-answering machines, programmes simulating concept formulation in humans, social behaviour etc.

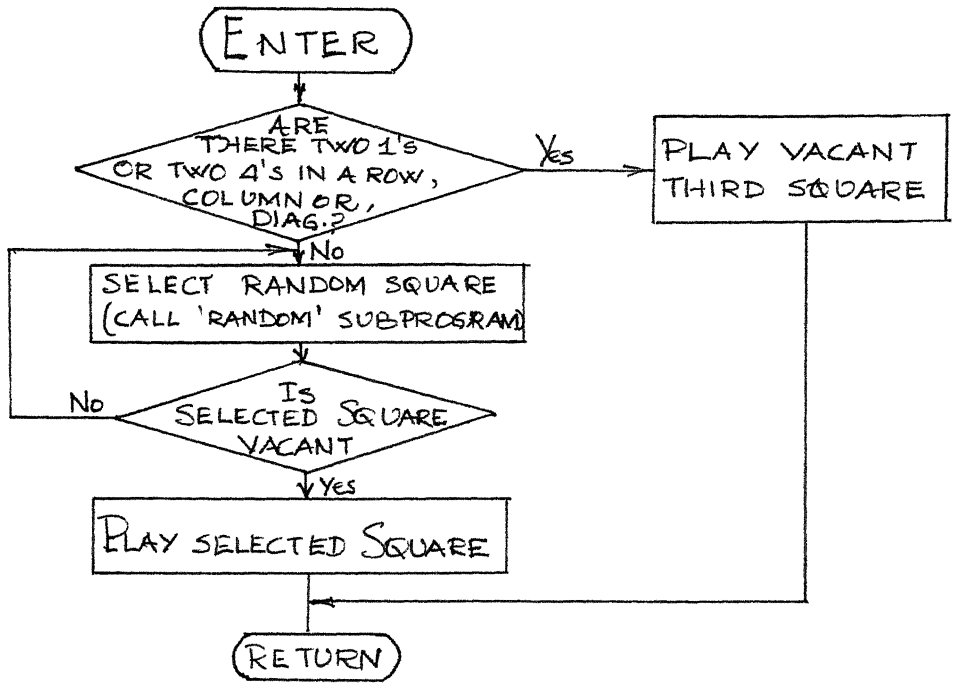
Since our main interest in this thesis has been in game-playing programmes, it seems a fitting end to quote Shannon's words regarding game playing machines and their human opponents: "On an equal time basis, the speed, patience and deadly accuracy of the machine would be telling against human fallibility".



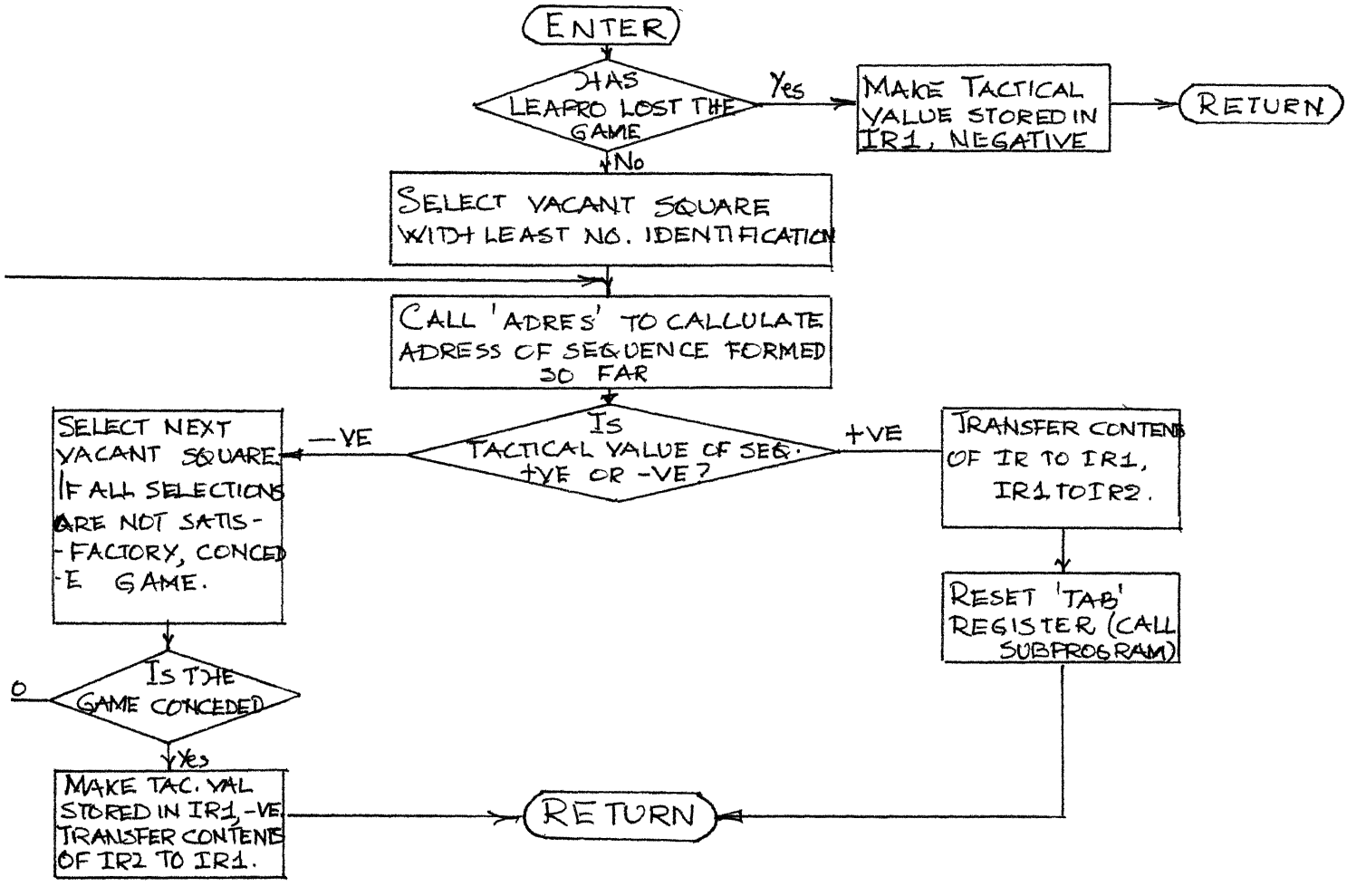
FLOW CHART 1. GAME BETWEEN TWO PROGRAMS IN THE SAME COMPUTER.



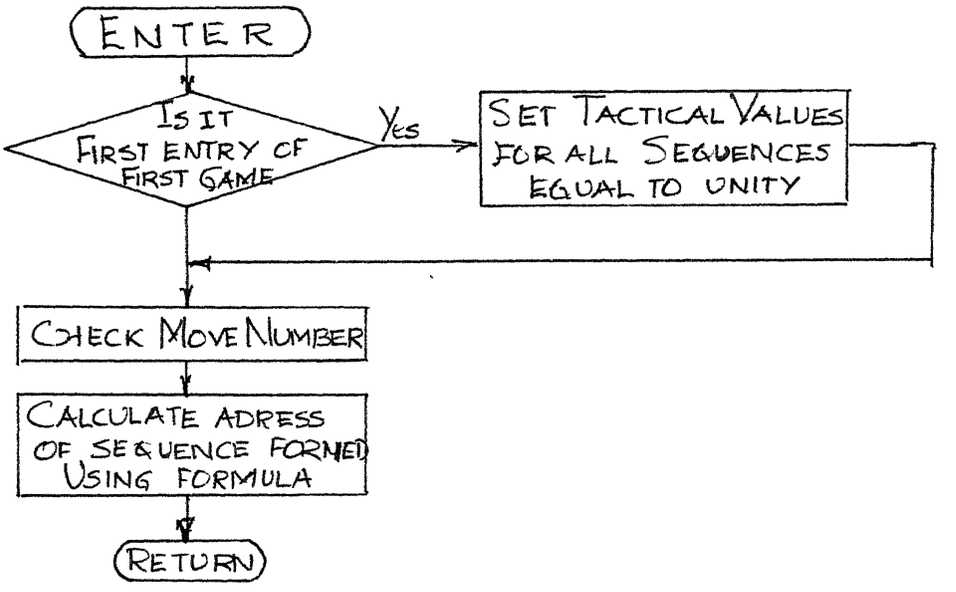
FLOW CHART II: 'EXECUTOR'



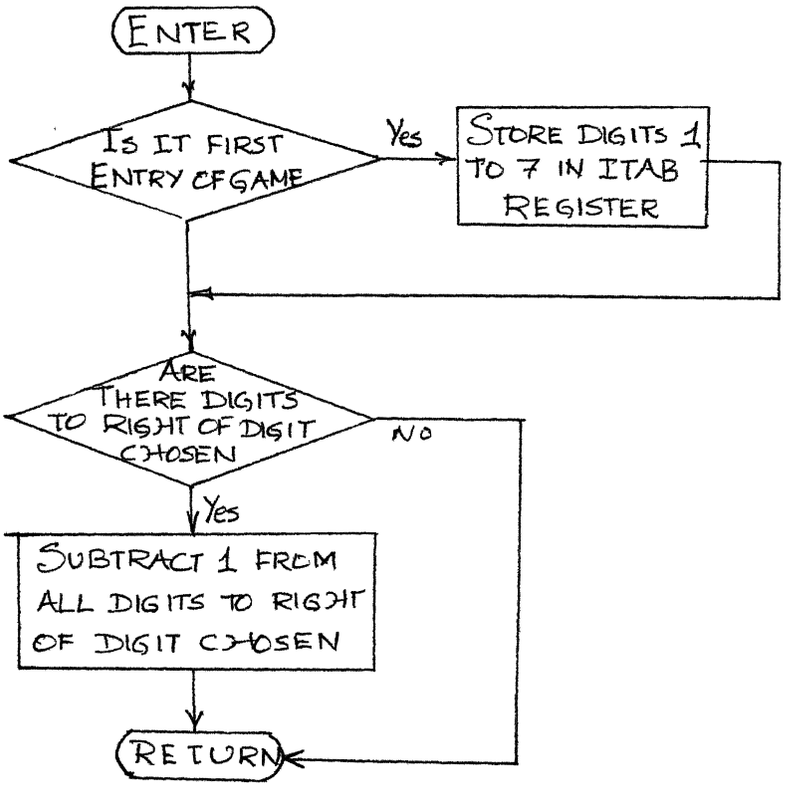
FLOW CHART III : 'INTPRC' SUBPROGRAM.



FLOW CHART IV : 'LEAPRO' SUBPROGRAM.



FLOW CHART V: 'ADRES' SUBPROGRAM



FLOW CHART VI: 'TAB' SUBPROGRAM

REFERENCES

1. Ashby, W.R. 1960, Design for a Brain, (Second Edition) John Wiley and Sons, New York.
2. Cannon, W.B. 1929, Quoted in The Brain as a Computer, by F.George, Pergamon Press, 1962.
3. Deese, 1952, Psychology of Learning, McGraw-Hill, New York.
4. Gelernter, 1963, in Computers and Thought, Feigenbaum and Feldman (Eds.), McGraw-Hill, New York.
5. Grey Walter, W. 1961, The Living Brain, Penguin Books, London.
6. Guthrie, E.R. 1935, The Psychology of Learning, Harper, New York.
7. Hilgard and Marquis, 1940, Conditioning and Learning, Appleton-Century-Crofts, New York.
8. Hormann, A.M. 1962, Programmes for Machine Learning, Part I, Information and Control, 5 (4).
9. ----------, 1964, Programmes for Machine Learning, Part II, Information and Control, 7 (1). Also see: Hormann: How a Computer System can Learn, IEEE Spectrum, July 1964.
10. Humphrey, 1933, Quoted in The Brain as a Computer, by F.George, Pergamon Press, 1962.
11. Newell, 1960, A Variety of Intelligent Learning in a General Problem Solver. In M.C.Yovits & S.Cameron, (Eds.), Self-Organising Systems, Pergamon Press, London.
12. Newell, 1963, in The Modeling of Mind, Edited by Sayre and Crosson, University of Notre Dame Press.

13. Newell, Shaw and Simon, 1963, in Computers and Thought, Feigenbaum and Feldman (Eds.), McGraw-Hill, New York.

14. Oettinger, A.E. 1952, Programming a digital computer to learn, Philosophical Magazine, 43: 1243-1262.

15. Samuel, A.L. 1963 in Computers and Thought, Feigenbaum and Feldman (Eds.), MsGraw-Hill, New York.

16. Shannon, C.E. 1950, Programming a Computer to play Chess, Philosophical Magazine, (March 1950), 256-275.

17. Thorpe, 1950, in 10.

18. Tonge, F. 1963 in ## 13.